

# Experiences Using a Meta-Data Based Integration Infrastructure

*T. Critchlow, R. Musick and T. Slezak*

This article was submitted to  
2<sup>nd</sup> International Workshop on Biomolecular Informatics  
Atlantic City, NJ  
February 27- March 2, 2000

**July 8, 1999**

U.S. Department of Energy

Lawrence  
Livermore  
National  
Laboratory

## DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

This report has been reproduced  
directly from the best available copy.

Available to DOE and DOE contractors from the  
Office of Scientific and Technical Information  
P.O. Box 62, Oak Ridge, TN 37831  
Prices available from (423) 576-8401  
<http://apollo.osti.gov/bridge/>

Available to the public from the  
National Technical Information Service  
U.S. Department of Commerce  
5285 Port Royal Rd.,  
Springfield, VA 22161  
<http://www.ntis.gov/>

OR

Lawrence Livermore National Laboratory  
Technical Information Department's Digital Library  
<http://www.llnl.gov/tid/Library.html>

# **Experiences Using a Meta-Data Based Integration Infrastructure**

**Terence Critchlow, Ron Musick, Tom Slezak**  
**Lawrence Livermore National Laboratory (LLNL)**

## **1. Introduction**

One of the classic challenges in bio-informatics has been the integration of multiple community data sources into a single, consistent data repository. Over the years, there have been several attempts to address this problem. So far, none have been entirely successful. The task is a daunting one: integrate large amounts of complex data, obtained from autonomous, heterogeneous, distributed data sources which change their data representation regularly, and present it in a consistent and intuitive way. On the surface, this challenge seems similar to the one faced by industry, and successfully addressed by data warehousing technology. Unfortunately, the major difference between commercial and scientific data – the dynamic nature of the data sources – makes directly applying warehousing technology infeasible. The primary reason is the significant amount of work required to update the warehouse so it can incorporate new data whenever a source changes its data representation. A related reason is the constant desire to add new data sources to the warehouse. This arises from both the scientist's ongoing quest to obtain more information about the data, and the increase in the number of relevant data sources being made publicly available. Because of these requirements, a direct application of data warehousing techniques to genomics sources results in a system that is too costly to maintain for an extended period of time. Thus the trend of new resources being developed, but quickly becoming obsolete and being ignored.

One approach to reducing these costs is to utilize meta-data to automatically generate components of the warehouse infrastructure. This approach, however, requires several difficult questions to be answered: what do we need to represent? What format should we use? How do we translate the meta-data into usable components? and, most importantly, does using meta-data really save any time or effort? In this paper, we discuss our experiences using the meta-data infrastructure developed as part of the DataFoundry project at LLNL. First, we present the DataFoundry architecture, including both the meta-data representation and how it is used, in Section 2. Section 3 describes our experiences using the meta-data infrastructure to integrate two genomic data sources into our warehouse, outlines some of the problems initially encountered, and contrasts the process to the traditional, manual approach. Finally, we briefly summarize the current status and future directions of the project.

## **2. Architecture**

DataFoundry is based on a mediated data warehouse architecture, shown in Figure 1, in which wrappers obtain data from the individual sources and pass it to a mediator. If a source is relational, the wrapper can easily obtain the data through SQL commands. However, if it is a flat file, as is often the case, the wrapper must first parse the file to obtain the data. The mediators transform the data from the source representation into the warehouse representation, and enter it into the warehouse. As shown in the figure, and discussed in Section 3, there may be multiple mediators populating the warehouse, and a mediator may be used by more than one wrapper. To reduce the cost of integrating new sources, DataFoundry automatically generates the mediators from meta-data.

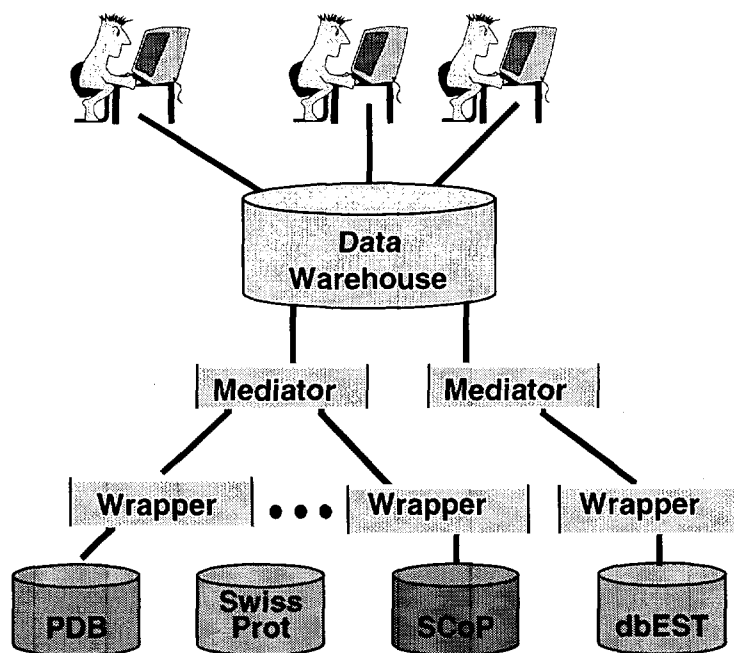


Figure 1 DataFoundry Architecture

DataFoundry uses meta-data to describe four concepts: *abstractions*, *transformations*, *schema*, and *mappings*. Abstractions define a class hierarchy reflecting the domain specific concepts contained in the data sources, including different attribute formats. For example, the *chain* abstraction has attributes length, name, one-character sequence and three-character sequence. Transformations define the set of legal translations between related attributes – for example between the one-character and three-character sequence attributes. Schema meta-data mirrors the target database (i.e. data warehouse) schema using an extended relational format. This meta-data is easily obtained from commercial RDBMSs, but has been extended to allow complex object definitions and comments to be included. Where appropriate, mappings define one-to-one correspondences between abstraction attributes and schema attributes. Currently, the meta-data is represented using Ontolingua, however it is in the process of being converted to an XML representation. We expect this new representation will allow us to leverage the generic resources, tools and interpreters developed by the XML community. For example, we may be able to obtain a generic graphical meta-data browser and editor, instead of having to develop a customized one.

We have implemented a mediator generator program that outputs both a mediator class and a translation library. The library defines a C++ class hierarchy, mirroring the abstraction hierarchy defined in the meta-data, but extended to include the appropriate constructors, destructors, get, and put methods. Because abstractions may define multi-values attributes, the put methods use linked list constructs where needed to mimic this capability. If the attribute requested through a get method is undefined, the method will attempt to obtain its value by transforming one of the other attributes, based on the transformations defined in the meta-data. This attempt may trigger multiple translations if the required attribute(s) is undefined. The mediator class defines methods that take high-level objects (i.e. instances of select classes) and enters them into the database. This requires performing transformations when appropriate, iterating over instances of multi-valued attributes, and entering the data into the warehouse only if all of the required information has been provided.

### 3. Experiences

To date, we have integrated four genomic data sources into the DataFoundry warehouse: PDB, SWISS-PROT, SCoP, and dbEST. The first two were integrated before the meta-data infrastructure was implemented, and were thus done manually. This required writing stand-alone programs to parse the data source into an intermediate representation, perform data transformations and enter the data into the warehouse. Obviously, this is not scalable because the maintenance cost of updating these programs as the source formats change is very high. The meta-data infrastructure described in the previous section was used to integrate the SCoP and dbEST databases. This was faster and easier because the class definitions created by the mediator generator were used as the parser's internal data representation, and the warehouse population code was automatically generated.

However, during the integration of these two sources we encountered a few interesting problems with the initial implementation of the mediator generator. First, proper memory deallocation, while seemingly very mundane, proved to be more challenging than anticipated. The initial version of the mediator generator produced classes having several memory leaks in them. While this was harmless enough in the test data sets, it had a serious performance impact when loading millions of objects. The first problem with the original destructors was that they did not correctly deallocate multi-valued attributes. Specifically, because they did not differentiate between list elements and regular class pointers, they would only delete the first element in a list. This problem, once identified, was easily corrected by modifying the mediator generator to iterate through list elements appropriately. The second, more challenging, problem was to ensure that memory was only deallocated once. Multiple deallocations were occurring because the same object was referenced by multiple pointers within the same enclosing object. For example, in dbEST, the same person could be referenced both as an author and as a map submitter. To address this problem, the destructors were modified to record the memory locations of the object they freed, and to only release the memory once.

Second, integrating these sources demonstrated the need to arbitrarily partition the warehouse to reduce redundancy and improve performance. Initially, several tables contained attributes that were null or constant for all data from a specific source (i.e. the source didn't have any corresponding attributes so defaults were used). To reduce the space required for these tables, they were duplicated under a different name without these columns. However, this created a problem with the mediator generator since the initial implementation assumed that it would be mapping abstractions onto a single target schema. Obviously, if arbitrary partitioning is allowed, that is not the case. To handle multiple targets the mediator generator could either create more complex mediators that associated the appropriate target schema with individual wrappers, or it could define multiple mediators. Since a single mediator that recognized all target schemata would quickly become prohibitively complex, we choose to modify our overall architecture to handle multiple mediators, one for each target. This was a dramatic departure from our initial belief that a single mediator would be sufficient for all of the source wrappers, and required updating our meta-data repository to contain mapping and schema components for each target (note the abstraction and translation components do not depend on the target).

One of the problems that has not yet been completely addressed is how to propagate in-place updates of a data source to the warehouse. Since an in-place update modifies data that has already been entered into the warehouse, simply entering the new data into the warehouse would violate integrity constraints and cause user confusion. The easiest way to perform this update, for a small number of objects, is to delete the entire entry and insert the new data into the database. However, this requires being able to identify and remove all data referenced by only that entry.

**Table 1 Integration Results**

<b>Activity \ Integration Style</b>	<b>Manual</b>	<b>Meta-Data</b>	<b>Diff</b>	<b>% Diff</b>
Understanding SCoP	2.0	2.0	0.0	0%
Writing wrapper	4.5	2.5	2.0	45%
Modifying schema	0.5	0.5	0.0	0%
Writing mediator	4.0	N/A	4.0	--
Modifying meta-data	N/A	1.0	(1.0)	--
Total time (in days)	11.0	6.0	5.0	45%

This turns out to be a complicated task since some objects, such as authors, may be referenced by multiple entries, and determining whether the current entry is the only reference can be difficult. Currently, the delete method is manually defined, but we believe the meta-data contains sufficient information to automatically generate it, if sufficient time is invested updating the mediator generator.

Despite these difficulties, this approach has great promise. Table 1 compares the time required to integrate SCoP into our warehouse using our meta-data based approach with the time it would take to manually integrate it. As the table shows, our approach significantly reduced the time required to perform the integration. As expected, we saved a considerable amount of time because updating the meta-data was significantly easier than writing the mediator from scratch. A surprising result was the amount of time saved by using the class hierarchy from the translation library as the parser's internal representation instead of creating a customized definition. In effect, the time spent updating the meta-data could be viewed as being spent defining the class representation. However, since the meta-data definition is at a much higher level (i.e. no need to write destructors), and it builds upon the existing meta-data, it didn't take as long as defining the classes in C++ would have. While we have not performed a similar comparison for the integration of dbEST, in part because it is a significantly more complex data source, the overall time reduction would probably not be as significant as it was for SCoP (the overall time would be dominated by shared tasks such as understanding the domain, identifying the schema, defining the parser, etc.). We believe that there would still be significant savings for the portions of the process affected by the meta-data infrastructure.

#### **4. Conclusions**

A data warehouse that presents data from many of the genomics community data sources in a consistent, intuitive fashion has long been a goal of bioinformatics. Unfortunately, it is one of the goals that has not yet been achieved. One of the major problems encountered by previous attempts has been the high cost of creating and maintaining a warehouse in a dynamic environment.

In this abstract we have outlined a meta-data based approach to integrating data sources that begins to address this problem. We have used this infrastructure to successfully integrate new sources into an existing warehouse in substantially less time than would have traditionally been required – and the resulting mediators are more maintainable than the traditionally defined ones would have been. In the final paper, we will describe in greater detail both our architecture and our experiences using this framework. In particular, we will outline the new, XML based representation of the meta-data, describe how the mediator generator works, and highlight other potential uses for the meta-data.